SYS ADMIN

Git & GitHub

สำหรับนักพัฒนาแอป ยุค AI



ดร.อาณัติ รัตนถิรกุล

Git & GitHub

สำหรับนักพัฒนาแอปยุค AI

เขียนโดย

ดร.อาณัติ รัตนถิรกุล

SYSADMIN KNOWLEDGE

Knowledge Center for IT Expert

“เรียนรู้ระบบแบบเข้าใจง่าย ใช้งานได้จริง”

สงวนลิขสิทธิ์ตามพระราชบัญญัติลิขสิทธิ์ พ.ศ. 2537 และที่แก้ไขเพิ่มเติม (ฉบับปี 2565)
โดย ซิสแอดมินโนว์เลดจ์ ห้ามทำซ้ำ คัดลอก ดัดแปลง หรือเผยแพร่ไม่ว่าในรูปแบบใด ๆ
ไม่ว่าจะทั้งหมดหรือบางส่วน เว้นแต่ได้รับอนุญาตเป็นลายลักษณ์อักษร
ชื่อเว็บไซต์, ชื่อโปรแกรม และชื่อผลิตภัณฑ์ต่าง ๆ ที่ปรากฏหรือกล่าวในหนังสือ
เป็นลิขสิทธิ์ขององค์กรหรือบริษัทผู้ผลิตนั้น ๆ

Git & GitHub สำหรับนักพัฒนาแอปยุค AI

พิมพ์ครั้งที่ 1: ตุลาคม 2569
โดย ดร. อาณัติ รัตนถิรกุล

ราคา 449.- บาท

พิสูจน์อักษร : ธนียา รัตนถิรกุล
หมวด: Programming

สำนักพิมพ์: ซิสแอดมิน โนว์เลดจ์
ISBN: 978-616-642-201-6

จัดทำโดย
ดร.อาณัติ รัตนถิรกุล

จัดจำหน่ายโดย
ซิสแอดมิน โนว์เลดจ์
เลขที่ 18 ซอยสุเหร่าหัวหมากน้อย แขวงหัวหมาก เขตบางกะปิ กรุงเทพฯ 10240
โทรศัพท์ 084-1629068 อีเมล book@sysadmin.in.th

กรณีต้องการซื้อเป็นจำนวนมาก เพื่อใช้ในการเรียนการสอน การฝึกอบรม หรือมอบเป็นของขวัญ
ในโอกาสพิเศษ กรุณาติดต่อเพื่อขอราคาพิเศษ โดยตรงได้ที่อีเมล book@sysadmin.in.th

คำนำ

ช่วงสองปีที่ผ่านมา คนที่เริ่มเขียนแอปครั้งแรกเปลี่ยนไปมาก นักศึกษาที่เพิ่งเรียน HTML ได้สองสัปดาห์สั่งผู้ช่วย AI สร้างเว็บที่มีระบบกรองสินค้าและตะกร้าได้ในบ่ายเดียว เจ้าของร้านที่ไม่เคยเขียนโค้ดเลยสร้างหน้าเมนูของตัวเองจากเครื่องมือสร้างแอปบนเว็บได้ ความเร็วในการได้โค้ดมาไม่ใช่ปัญหาอีกต่อไป

ปัญหาย้ายไปอยู่ที่ว่า เมื่อโค้ดเปลี่ยนเร็วขนาดนี้ เรารู้ได้อย่างไรว่าอะไรเปลี่ยนไปบ้าง ถ้าแอปที่เมื่อวานใช้ได้วันนี้พัง จะย้อนกลับอย่างไร และจะตรวจงานที่ AI ส่งมาได้อย่างไรก่อนนำไปใช้จริง คำถามเหล่านี้ไม่ใช่เรื่องใหม่ นักพัฒนาซอฟต์แวร์ตอบมันด้วย Git มากว่ายี่สิบปีแล้ว สิ่งที่ใหม่คือคนที่ต้องตอบคำถามนี้ไม่ได้มีแค่นักพัฒนามืออาชีพอีกต่อไป หนังสือภาษาไทยที่สอนเรื่อง Git & Github ที่มีอยู่ส่วนใหญ่เขียนไว้หลายปีก่อน เน้นนักพัฒนาที่เขียนโค้ดเองทุกบรรทัด และบางเล่มยังใช้บริการที่ไม่ได้เป็นที่นิยมแล้ว เล่มนี้เขียนขึ้นสำหรับผู้เริ่มต้นพัฒนาแอปในปัจจุบัน ทั้งคนที่เขียนโค้ดเองและคนที่ใช้ AI ช่วยเขียน โดยยึดหลักเดียวตลอดเล่ม คือตรวจก่อนใช้ ดูสถานะ อ่านความเปลี่ยนแปลง ทดสอบ แล้วจึงบันทึกหรือรวมงาน ไม่ว่าโค้ดชิ้นนั้นใครเขียน

ทุกบทสอนผ่านการลงมือทำกับแอปตัวเดียวกัน คือหน้าเว็บเมนูของร้านกาแฟสมมติ (ผู้อ่านสามารถเปลี่ยนเป็นแอปที่ต้องการได้) เริ่มจากสามไฟล์ในโฟลเดอร์ธรรมดา ไปจนถึงแอปออนไลน์ที่มีชุดทดสอบอัตโนมัติและออกเวอร์ชันส่งมอบได้ คำสั่งทุกคำสั่งและผลลัพธ์ที่ยกมาในเล่มได้มาจากการรันจริง ส่วนที่ทำงานกับ GitHub ทดสอบบนบัญชีจริงเมื่อเดือนตุลาคม 2569 และ repo ที่ใช้ทดสอบเปิดให้ทุกคนเข้าไปดูได้ตามที่บอกไว้ในหน้าวิธีใช้หนังสือ

เครื่องมือ AI และหน้าจอของ GitHub เปลี่ยนเร็ว ชื่อปุ่มหรือราคาบางอย่างในเล่มอาจไม่ตรงกับที่ผู้อ่านเห็นในวันที่อ่าน เล่มนี้จึงใช้คำสั่งของ Git เป็นแกน ซึ่งเปลี่ยนช้ามาก และอธิบายหลักการก่อนหน้าจอทุกครั้ง ถ้าผู้อ่านพบจุดที่ทำตามแล้วไม่ได้ผลตามที่เขียนไว้ แจ้งได้ที่ Issues ของ repo ตัวอย่างประจำเล่ม จะได้ทำการปรับปรุงแก้ไขในฉบับต่อไป

ดร.อาณัติ รัตนถิรกุล
ซิสแอดมิน โนว์เลดจ์
ตุลาคม 2569

วิธีใช้หนังสือเล่มนี้

ใครควรอ่าน

เล่มนี้เขียนสำหรับผู้เริ่มต้นพัฒนาแอปที่ยังไม่เคยใช้ Git จริงจัง ทั้งนักศึกษา คนที่เรียนเขียนเว็บด้วยตัวเอง และคนที่สร้างแอปด้วยผู้ช่วย AI อย่าง GitHub Copilot, Cursor, Claude Code, ChatGPT Codex, Google Antigravity, Lovable หรือ Bolt พื้นฐานที่ต้องมีมีเพียงเปิดโฟลเดอร์ใน VS Code ได้ และพิมพ์คำสั่งใน terminal ตามตัวอย่างได้ ไม่จำเป็นต้องเขียนโค้ดเองเป็น

อ่านตามลำดับ

ทุกบทต่อยอดจากบทก่อนหน้า และทำงานกับแอปตัวเดียวกันต่อเนื่องทั้งเล่ม จึงควรอ่านตามลำดับ เล่มแบ่งเป็น 4 ภาค ดังนี้

1. **ภาค 1** บทที่ 1 ถึง 4 เริ่มต้นบันทึกและตรวจสอบโค้ดในเครื่อง
2. **ภาค 2** บทที่ 5 ถึง 7 แก้พลาด ย้อนเวลา และทดลองฟีเจอร์บน branch
3. **ภาค 3** บทที่ 8 ถึง 10 ขึ้น GitHub ทำงานกับ AI และรีวิวงานด้วย Pull Request
4. **ภาค 4** บทที่ 11 และ 12 ตรวจอัตโนมัติ เผยแพร่แอป และส่งมอบเป็นเวอร์ชัน

บทที่ 1 ถึง 7 ใช้แค่ Git ในเครื่อง ไม่ต้องมีบัญชี GitHub คนที่ใช้ Git อยู่แล้วบ้างอาจอ่านบทที่ 1 ถึง 4 เร็ว ๆ แต่อย่าข้ามบทที่ 5 ซึ่งเป็นพื้นฐานของการแก้ปัญหาทั้งเล่ม

ไฟล์ประกอบและ repo ตัวอย่างประจำเล่ม

ไฟล์ตั้งต้นและเฉลยของทุกบทอยู่ใน repo ตัวอย่างประจำเล่มบน GitHub

https://github.com/arnutkmitl/coffee-menu-test

ดาวน์โหลดทั้งหมดเป็นไฟล์ zip ได้โดยไม่ต้องมีบัญชี GitHub และไม่ต้องรู้จัก Git มาก่อน

```
https://github.com/arnutkmitl/coffee-menu-test/archive/refs/heads/main.zip
```

แตกไฟล์แล้วจะได้โฟลเดอร์ coffee-menu-test-main ไฟล์ประกอบของแต่ละบทอยู่ในโฟลเดอร์ย่อย book-files แยกตามบท เช่น ch02-start คือแอปตั้งต้นของบทที่ 2 และ ch05-end คือสภาพของโปรเจกต์หลังจบบทที่ 5 รายละเอียดของทุกโฟลเดอร์อยู่ในภาคผนวก ช

repo เดียวกันนี้คือโปรเจกต์ที่ผู้เขียนใช้ทดสอบทุกขั้นตอนในบทที่ 8 ถึง 12 บนบัญชีจริง เปิดดูประวัติ commit, Issue, Pull Request, ผลการรันของ GitHub Actions และ release ได้ทั้งหมด ใช้เทียบกับงานของตัวเองได้เมื่อไม่แน่ใจว่าทำถูกหรือไม่ และใช้เป็น repo สำหรับฝึก fork ในบทที่ 12

2 เส้นทาง: เขียนเองหรือให้ AI เขียน

แบบฝึกที่ต้องแก้โค้ดมีให้เลือกสองทาง ทางแก้เองมีโค้ดให้พิมพ์หรือคัดลอก ทางใช้ AI มี prompt ภาษาไทยที่ระบุไฟล์และขอบเขตไว้แล้ว ใช้ผู้ช่วย AI แบบฟรีตัวใดก็ได้ ทั้งสองทางต้องได้แอปที่ทำงานตามเกณฑ์เดียวกัน แต่โค้ดและรหัส commit ไม่จำเป็นต้องเหมือนในหนังสือ ใครไม่อยากใช้ AI เลยก็ทำได้ครบทุกบท บทที่ต้องตรวจงานของ AI มีไฟล์จำลองงานที่ AI ส่งกลับมาให้แล้ว

รูปแบบในเล่ม

คำสั่งที่ให้พิมพ์และผลลัพธ์ที่ควรเห็นแสดงในกรอบตัวอักษรแบบเครื่องพิมพ์ดีด ผลลัพธ์ในเครื่องของผู้อ่านจะต่างจากในหนังสือในบางส่วนเสมอ เช่น รหัส commit ชื่อผู้ใช้ วันเวลา และตัวเลขแสดงความเร็ว ให้ดูที่โครงสร้างและข้อความ

หลักเป็นสำคัญ และกล่องหมายเหตุให้ความรู้เพิ่มเติม (สีฟ้าอ่อน) กล่องข้อควรระวังเตือนจุดที่ทำให้งานหายหรือข้อมูลรั่วได้ (สีเหลืองอ่อน) กล่องเคล็ดลับที่ขึ้นต้นด้วยคำว่า ตรวจก่อนใช้ คือขั้นตอนของวงจรตรวจงานที่เพิ่มความละเอียดขึ้นทีละบท (สีเขียวอ่อน) และกล่อง Workshop คือแบบฝึกท้ายบทที่ควรทำจริงก่อนอ่านบทถัดไป (สีม่วงอ่อน)

เครื่องมือที่ใช้

เล่มนี้ทดสอบกับ Git 2.50 ถึง 2.55, VS Code รุ่นปัจจุบัน, Git GUI Clients, GitHub CLI 2.102 และ Node.js 24 บน Windows 11 คำสั่งของ Git ทำงานเหมือนกันบน macOS และ Linux จุดที่ต่างกันระหว่างระบบปฏิบัติการมีบอกไว้ในเนื้อหา เครื่องมือทั้งหมดใช้ได้ฟรี บริการของ GitHub ที่ใช้ในเล่ม ได้แก่ Actions และ Pages ใช้ได้ฟรีกับ repo แบบ public บนแผน Free ณ วันที่ทดสอบ

ดาวน์โหลดได้ที่

- Git for Windows : https://git-scm.com/install/windows
- Git for Mac : https://git-scm.com/install/mac
- Git for Linux: https://git-scm.com/install/linux
- Git GUI Clients: https://git-scm.com/tools/guis
- Git CLI: https://cli.github.com
- NodeJS: https://nodejs.org/en/download
- VSCode: https://code.visualstudio.com
- GitHub: https://github.com

สารบัญ

เรื่อง	หน้า
คำนำ	ข
วิธีใช้หนังสือเล่มนี้	ง
ภาค 1 เริ่มต้นบันทึกและตรวจสอบโค้ด	
บทที่ 1 ทำไมนักพัฒนาแอปยุค AI ต้องมี Git	**1**
1.1 เมื่อแอปที่เคยใช้ได้กลับพัง	2
1.2 Git คือจุดเซฟของโปรเจกต์	4
1.3 Git ในเครื่องกับ GitHub บนออนไลน์	6
1.4 Repository, commit และประวัติของโปรเจกต์	7
1.5 โค้ดที่เขียนเองและโค้ดจาก AI ต้องตรวจอะไรบ้าง	8
1.6 แอปเมนูร้านกาแฟและเส้นทางของเล่มน	9
Workshop สืบหาสาเหตุที่เมนูหาย	11
สรุปท้ายบท	14
บทที่ 2 ติดตั้ง Git และ VS Code แล้วเปิดโปรเจกต์แรก	**15**
2.1 ไฟล์ โฟลเดอร์ และตำแหน่งของ terminal	16
2.2 ติดตั้ง Git และ VS Code	18
2.3 ตรวจการติดตั้ง	19
2.4 ตั้งค่าชื่อ อีเมล และชื่อ branch เริ่มต้น	20
2.5 ชื่อผู้บันทึก commit ไม่ใช่การล็อกอิน	24

เรื่อง	หน้า
2.6 เปิดแอปเมนูร้านกาแฟครั้งแรก	25
2.7 ติดตั้ง VSCode พร้อมกำหนดค่าพื้นฐาน	27
Workshop เช็กลิสต์เครื่องพร้อม	30
สรุปท้ายบท	33
บทที่ 3 Commit แรก: จุดเซฟของโปรเจกต์	**34**
3.1 สามพื้นที่ที่ Git ใช้	35
3.2 สร้าง repository ด้วย git init	36
3.3 อ่านสถานะด้วย git status	38
3.4 เลือกไฟล์ด้วย git add	39
3.5 บันทึกด้วย git commit	41
3.6 เขียนข้อความ commit ให้สื่อ	44
3.7 อ่านประวัติด้วย git log	45
3.8 ทำแบบเดียวกันผ่าน VS Code	46
Workshop สามจุดเซฟแรกของบ้านสวนคาเฟ่	48
สรุปท้ายบท	55
บทที่ 4 ตรวจว่าโค้ดเปลี่ยนอะไร: diff, log และ .gitignore	**56**
4.1 อ่าน diff ทีละส่วน	57
4.2 diff ก่อนและหลัง git add	60
4.3 ดู diff ใน VS Code	63
4.4 จับการแก้เกินขอบเขต	64

เรื่อง	หน้า
4.5 ย้อนดูประวัติ: git show, git log และ git blame	65
4.6 .gitignore: กันไฟล์ที่ไม่ควรเข้า repo	67
4.7 เมื่อไฟล์ถูกติดตามไปแล้ว	72
Workshop เปลี่ยนธีมโดยไม่ให้ราคาเปลี่ยนตาม	74
สรุปท้ายบท	78
ภาค 2 แก้พลาดและทดลองฟีเจอร์	
บทที่ 5 ย้อนเวลาและกู้งานที่พัง	**80**
5.1 พังแบบไหน ใช้คำสั่งไหน	81
5.2 เอาไฟล์ออกจาก staging area	83
5.3 ทิ้งการแก้ไขทั้งไฟล์	84
5.4 แก้ commit ล่าสุดด้วย --amend	86
5.5 ย้อนผลของ commit ด้วย git revert	87
5.6 เปิดดูโค้ดเวอร์ชันเก่า	90
5.7 ห้องทดลอง: git reset และ git reflog	91
Workshop 5 สถานการณ์จำลองระบบพัง	96
สรุปท้ายบท	102
บทที่ 6 Branch: ลองไอเดียใหม่โดยไม่ทำของเดิมพัง	**103**
6.1 branch คือป้ายชี้ commit	104
6.2 ตรวจงานค้างก่อนสร้างหรือสลับ branch	106
6.3 สร้าง branch ด้วย git switch -c	106
6.4 พัฒนาฟีเจอร์เป็น commit ขนาดเล็ก	107

เรื่อง	หน้า
6.5 เทียบ branch ก่อนรวม	110
6.6 รวมด้วย git merge	112
6.7 พักงานด้วย stash และทิ้ง branch ที่ไม่ใช้	115
Workshop ธีมมืด ไอเดียที่ทิ้ง และงานด่วนระหว่างทาง	118
สรุปท้ายบท	122
บทที่ 7 Merge conflict ไม่น่ากลัว	**123**
7.1 ทำไม Git รวมบางจุดให้ไม่ได้	124
7.2 สร้าง conflict ที่ทำซ้ำได้	125
7.3 อ่าน conflict	127
7.4 ตัดสินจากโจทย์ ไม่ใช่จากปุ่ม	129
7.5 ตรวจผล แล้วทำ merge ให้เสร็จ	132
7.6 ยกเลิก merge ด้วย git merge --abort	134
7.7 ให้ AI ช่วยแก้ conflict	135
7.8 ลดโอกาสเกิด conflict	136
Workshop ลาเต้สองฝั่ง	137
สรุปท้ายบท	139
ภาค 3 เชื่อม GitHub และทำงานร่วมกับ AI	
บทที่ 8 ขึ้น GitHub: push, pull และ clone	**141**
8.1 เตรียมบัญชีและเชื่อมเครื่องกับ GitHub	142
8.2 public หรือ private	148
8.3 สร้าง repo บน GitHub และเชื่อม origin	149
8.4 ส่งงานด้วย push	151

เรื่อง	หน้า
8.5 clone: เริ่มงานจากสำเนาบน GitHub	152
8.6 fetch และ pull: รับงานจาก GitHub	153
8.7 push ถูกปฏิเสธ	156
8.8 README ที่คนอื่นใช้งานได้	158
Workshop สองสำเนาหนึ่ง repo	161
สรุปท้ายบท	165
บทที่ 9 ใช้ Git ร่วมกับผู้ช่วยเขียนโค้ด AI	**166**
9.1 ผู้ช่วย AI สองแบบ	167
9.2 ก่อนสั่งงาน: ตรวจ repo และสร้าง branch	169
9.3 เขียนโจทย์ที่ตรวจได้	170
9.4 ไฟล์กติกาของโปรเจกต์	171
9.5 อ่าน diff เทียบกับคำอธิบายของ AI	173
9.6 เก็บส่วนที่ดี ทิ้งส่วนที่เกิน	175
9.7 commit ที่ AI สร้างให้	177
9.8 ความลับและสิ่งที่ต้องทำเมื่อรั่ว	178
Workshop ตัวกรองเมนูที่ตรวจได้ทุกขั้น	180
สรุปท้ายบท	184
บทที่ 10 Pull Request และการรีวิวโค้ดก่อนรวม	**185**
10.1 เขียน Issue: งานที่ตรวจรับได้	186
10.2 ทำงานบน branch จาก main ล่าสุด	189
10.3 push branch และเปิด Pull Request	190
10.4 คำอธิบาย PR: สิ่งที่เปลี่ยนและวิธีทดสอบ	191

เรื่อง	หน้า
10.5 รีวิว: อ่าน Files changed	192
10.6 ส่ง commit เพิ่มเพื่อตอบผลรีวิว	194
10.7 เลือกวิธีรวมและเก็บกวาด	195
10.8 กฎของ repo และ PR ที่ AI เปิดให้	198
Workshop จาก Issue สู่ main	200
สรุปท้ายบท	203
ภาค 4 ตรวจอัตโนมัติ เผยแพร่ และส่งมอบ	
บทที่ 11 GitHub Pages และ Actions: แอปขึ้นออนไลน์ฟรี	**205**
11.1 อะไรตรวจอัตโนมัติได้ อะไรยังต้องตรวจเอง	206
11.2 แยกตรรกะออกมาให้ทดสอบได้	207
11.3 ชุดทดสอบ	210
11.4 workflow: สั่งให้ GitHub รันทดสอบ	212
11.5 รันครั้งแรกผ่าน PR	215
11.6 ทำให้เทสแดงโดยตั้งใจ	220
11.7 เผยแพร่ด้วย GitHub Pages	224
11.8 การสร้าง GitHub Pages	225
11.9 โควตา ความเป็นสาธารณะ และความลับใน Actions	227
Workshop แดง เขียว แล้วออนไลน์	228
สรุปท้ายบท	232

เรื่อง	หน้า
บทที่ 12 Workflow ประจำวันและก้าวสู่การทำงานเป็นทีม	**233**
12.1 workflow ประจำวัน	234
12.2 รับโจทย์และกำหนดเกณฑ์รับงาน	236
12.3 พัฒนาเป็น commit ที่อ่านรู้เรื่อง	237
12.4 ตรวจ ทดสอบ และเปิด PR	239
12.5 รีวิวและส่งงานรอบใหม่	240
12.6 รวมงานและออกเวอร์ชัน	241
12.7 บั๊กหลังส่งมอบ	244
12.8 ทำงานร่วมกับคนอื่น	247
Workshop ส่งมอบตะกร้าสาธิต	251
สรุปท้ายบท	255
ภาคผนวก	
ภาคผนวก ก คำสั่งตามสถานการณ์	257
ภาคผนวก ข พังแบบไหน ใช้คำสั่งไหน	263
ภาคผนวก ค ศัพท์ Git และ GitHub	267
ภาคผนวก ง แบบตรวจก่อน commit, push, merge และเผยแพร่	271
ภาคผนวก จ ตัวอย่าง prompt ภาษาไทยสำหรับผู้ช่วย AI	273
ภาคผนวก ฉ แก้ปัญหาที่เจอบ่อย	276
ภาคผนวก ช วิธีใช้ไฟล์ประกอบและเฉลย	281
เกี่ยวกับผู้เขียน	284

ภาค 1

เริ่มต้นบันทึกและตรวจสอบโค้ด



บทที่ 1

ทำไมนักพัฒนาแอปยุค AI ต้องมี Git

1.1 เมื่อแอปที่เคยใช้ได้กลับพัง
1.2 Git คือจุดเซฟของโปรเจกต์
1.3 Git ในเครื่องกับ GitHub บนออนไลน์
1.4 Repository, commit และประวัติของโปรเจกต์
1.5 โค้ดที่เขียนเองและโค้ดจาก AI ต้องตรวจอะไรบ้าง
1.6 แอปเมนูร้านกาแฟและเส้นทางของเล่มน
Workshop สืบหาสาเหตุที่เมนูหาย



โปรเจกต์แรกของมือใหม่แทบทุกคนมีโฟลเดอร์หน้าตาแบบนี้ app, app_backup, app_final, app_final_v2, app_final_จริง ทุกโฟลเดอร์คือความพยายามเก็บ "เวอร์ชันที่ยังใช้ได้" เอาไว้ก่อนลงมือแก้รอบใหม่ วิธีนี้ใช้ได้อยู่พักหนึ่ง จนวันที่แอปพังแล้วต้องตอบคำถามว่าโฟลเดอร์ไหนคือเวอร์ชันที่ดีล่าสุด และสองโฟลเดอร์นั้นต่างกันตรงไหน คำถามแรกยังพอเดาได้จากวันที่ คำถามที่สองแทบไม่มีทางตอบได้ถ้าไม่เปิดไฟล์ไล่ดูทีละบรรทัด

ปัญหานี้หนักขึ้นเมื่อเราใช้ AI ช่วยเขียนโค้ด คำสั่งเดียวอย่าง "ช่วยปรับหน้าเมนูให้สวยขึ้น" อาจทำให้ AI แก้ไฟล์สามสี่ไฟล์พร้อมกันภายในไม่กี่วินาที ถ้าผลออกมาดีก็สบาย แต่ถ้าปุ่มหนึ่งหยุดทำงาน เรามักไม่รู้ด้วยซ้ำว่า AI แตะไฟล์ไหนไปบ้าง

บทนี้ยังไม่ต้องติดตั้งอะไร เป้าหมายคือเห็นว่า Git แก้ปัญหาอะไร Git กับ GitHub ต่างกันอย่างไร และทำไมหนังสือเล่มนี้ถึงย้ำเรื่อง "ตรวจก่อนใช้" ตั้งแต่บทแรกจนบทสุดท้าย

1.1 เมื่อแอปที่เคยใช้ได้กลับพัง

ลองดูสองเหตุการณ์ที่เกิดกับแอปตัวเดียวกัน คือหน้าเว็บเมนูของร้านกาแฟเล็ก ๆ ที่มีรายการเครื่องดื่ม ราคา และปุ่มเรียงตามราคา

เหตุการณ์แรก นักศึกษาคนหนึ่งเขียนโค้ดเองจนดึก ตั้งใจจะเปลี่ยนสีหัวข้อร้านจากสีน้ำตาลเป็นสีเขียว ระหว่างนั้นเห็นว่าชื่อฟังก์ชัน sortByPrice ไม่ค่อยสื่อ จึงเปลี่ยนเป็น sortMenuByPrice ไปด้วย กดบันทึก รีเฟรชเบราว์เซอร์ หัวข้อเป็นสี

เขียวตามต้องการ แต่รายการเมนูทั้งหมดหายไปจากหน้า เขาจำได้ว่าแก้ไปสองจุด แต่ไม่แน่ใจว่ามีจุดอื่นอีกหรือเปล่า

เหตุการณ์ที่สอง อีกคนหนึ่งไม่ได้เขียนโค้ดเอง เขาพิมพ์ในหน้าต่างแชตของผู้ช่วย AI ว่า "เปลี่ยนสีชื่อร้านเป็นสีเขียว" AI ตอบกลับว่าแก้เรียบร้อย พร้อมสรุปสั้น ๆ ว่าปรับสีใน style.css ผลที่เห็นในเบราว์เซอร์เหมือนเหตุการณ์แรกทุกอย่าง หัวข้อเป็นสีเขียว แต่เมนูหาย สิ่งที่ AI ไม่ได้บอกคือมันเปลี่ยนชื่อฟังก์ชันใน app.js ไปด้วย "เพื่อให้โค้ดอ่านง่ายขึ้น"

สองเหตุการณ์นี้มีต้นเหตุเดียวกัน ชื่อฟังก์ชันถูกเปลี่ยนในบรรทัดที่ประกาศฟังก์ชัน แต่บรรทัดท้ายไฟล์ที่ผูกปุ่มเข้ากับฟังก์ชันยังเรียกชื่อเดิม เมื่อเบราว์เซอร์อ่านถึงบรรทัดนั้นแล้วหาฟังก์ชันชื่อ sortByPrice ไม่เจอ สคริปต์จึงหยุดทำงานก่อนถึงคำสั่งที่แสดงรายการเมนู

ความต่างของสองเหตุการณ์อยู่ที่ว่าใครเป็นคนแก้เท่านั้น ส่วนคำถามที่ทั้งคู่ต้องตอบให้ได้เหมือนกันมีสามข้อ

1. ก่อนแอปพัง ไฟล์อยู่ในสภาพไหน
2. ระหว่างสภาพนั้นกับตอนนี้ มีอะไรเปลี่ยนไปบ้าง ทุกไฟล์ ทุกบรรทัด
3. จะเก็บส่วนที่ดี (สีเขียว) และทิ้งส่วนที่ทำให้พัง (ชื่อฟังก์ชัน) ได้อย่างไร

ถ้าไม่มีเครื่องมือช่วย สามคำถามนี้ตอบยาก โดยเฉพาะข้อสอง ทางที่คนส่วนใหญ่ทำคือกด Undo ในโปรแกรมแก้โค้ดย้อนไปเรื่อย ๆ ซึ่งใช้ได้เฉพาะไฟล์ที่ยังเปิดค้างอยู่ และ Undo แยกกันคนละไฟล์ ส่วนไฟล์ที่ AI แก้โดยเราไม่ได้เปิด จะไม่มีประวัติ Undo ให้เลย

1.2 Git คือจุดเซฟของโปรเจกต์

คนที่เคยเล่นเกมยาว ๆ คงรู้จักจุดเซฟ ก่อนเข้าห้องบอสเราเซฟเกมไว้ ถ้าแพ้ก็โหลดกลับมาที่จุดนั้นได้ ไม่ต้องเริ่มใหม่ตั้งแต่ด่านแรก Git ทำหน้าที่คล้ายกันกับโฟลเดอร์โปรเจกต์ เราสั่งให้ Git บันทึกสภาพของไฟล์ทั้งหมด ณ จังหวะที่เราพอใจ จุดบันทึกแต่ละจุดเรียกว่า commit

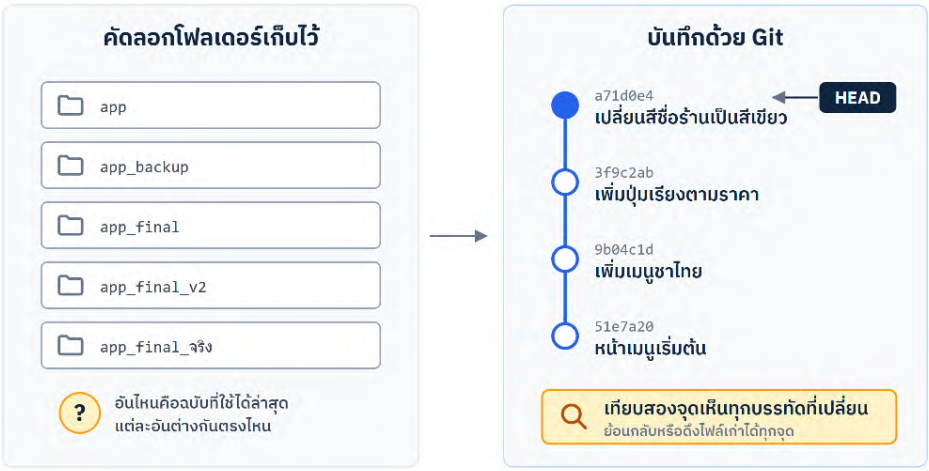
แต่ Git ทำได้มากกว่าจุดเซฟในเกมหลายเรื่อง

เรื่องแรก แต่ละ commit มีข้อความกำกับที่เราเขียนเอง เช่น "เพิ่มปุ่มเรียงตามราคา" หรือ "เปลี่ยนสีชื่อร้านเป็นสีเขียว" เวลาย้อนดูประวัติจึงอ่านได้ว่าโปรเจกต์เดินมาอย่างไร ไม่ต้องเดาจากชื่อโฟลเดอร์

เรื่องที่สอง Git เทียบความต่างระหว่างจุดใดก็ได้ให้เราดู ผลที่ได้เรียกว่า diff บอกละเอียดถึงระดับบรรทัดว่าไฟล์ไหนมีบรรทัดไหนถูกลบ บรรทัดไหนถูกเพิ่ม ถ้าสองคนในเหตุการณ์ข้างต้นมี commit ก่อนแก้ไว้ คำถามข้อสองจะได้คำตอบในคำสั่งเดียว และจะเห็นทันทีว่ามีการแก้ app.js ซึ่งไม่เกี่ยวกับเรื่องสีเลย

อีกเรื่องคือ Git ไม่ได้เก็บแค่จุดเซฟล่าสุด แต่เก็บทุก commit ต่อกันเป็นสาย เราย้อนกลับไปดูหรือดึงไฟล์จาก commit เก่าเมื่อไรก็ได้ ย้อนเฉพาะบางไฟล์ก็ได้ หรือสร้าง commit ใหม่ที่หักล้างผลของ commit ที่ทำให้พังโดยไม่ลบประวัติทิ้งก็ได้





ภาพที่ 1-1 การคัดลอกโฟลเดอร์เก็บไว้ เทียบกับการบันทึกประวัติด้วย Git

ข้อที่ต้องเข้าใจให้ตรงตั้งแต่ต้นคือการกด Save ในโปรแกรมแก้โค้ดไม่ใช่การ commit การกด Save เขียนเนื้อหาล่าสุดลงไฟล์ในดิสก์ ทับของเดิม ไม่มีประวัติ ส่วนการ commit คือการบอก Git ว่า "สภาพไฟล์ตอนนี้คือจุดที่อยากจำไว้" เราจะ Save เป็นร้อยครั้งระหว่างทำงาน แต่ commit เฉพาะตอนที่งานชิ้นหนึ่งเสร็จและผ่านการตรวจแล้ว

หมายเหตุ

ระบบที่เก็บประวัติการเปลี่ยนแปลงของไฟล์แบบนี้เรียกรวม ๆ ว่า version control Git เป็น version control ที่นิยมที่สุดในปัจจุบัน สร้างขึ้นในปี 2005 เพื่อใช้พัฒนา Linux kernel และเป็นซอฟต์แวร์โอเพนซอร์สที่ใช้ฟรี

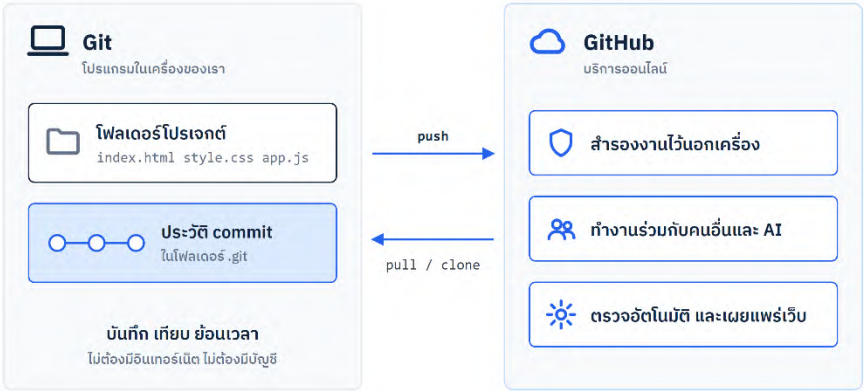
1.3 Git ในเครื่องกับ GitHub บนออนไลน์

มือใหม่จำนวนมากเรียก Git กับ GitHub สลับกันเหมือนเป็นสิ่งเดียวกัน ความจริงเป็นของสองอย่างที่ทำงานคู่กัน

Git เป็นโปรแกรมที่ติดตั้งลงเครื่องของเรา ทำงานกับโฟลเดอร์ในเครื่องโดยตรง ไม่ต้องมีอินเทอร์เน็ต ไม่ต้องมีบัญชีผู้ใช้ที่ไหน การ commit การดู diff และการย้อนเวลาทั้งหมดเกิดขึ้นในเครื่อง บทที่ 2 ถึงบทที่ 7 ของหนังสือเล่มนี้ใช้ Git อย่างเดียว ไม่แตะ GitHub เลย

GitHub เป็นบริการออนไลน์สำหรับฝากโปรเจกต์ที่ใช้ Git ไว้บนอินเทอร์เน็ต เราใช้มันเป็นที่สำรองงานนอกเครื่อง เป็นที่ทำงานร่วมกับคนอื่น และใช้เครื่องมือเสริมที่ GitHub มีให้ เช่น หน้าเว็บสำหรับรีวิวโค้ดก่อนรวมงาน ระบบตรวจโค้ดอัตโนมัติ และบริการเผยแพร่เว็บไซต์ฟรี เราจะเริ่มใช้ GitHub ในบทที่ 8

ลองนึกภาพแบบนี้ Git คือสมุดบันทึกที่อยู่ในกระเป๋าเรา GitHub คือตู้เก็บเอกสารกลางที่เราถ่ายสำเนาสมุดไปฝากไว้ และเพื่อนในทีมก็ถ่ายสำเนาของเขาไปฝากด้วย สมุดในกระเป๋าใช้ได้แม้ไม่มีตู้ แต่ตู้ไม่มีประโยชน์ถ้าไม่มีใครเขียนสมุด



ภาพที่ 1-2 หน้าที่ของ Git ในเครื่อง และ GitHub บนออนไลน์

หมายเหตุ

GitHub ไม่ใช่บริการเดียวที่ทำหน้าที่นี้ ยังมี GitLab และ Bitbucket ซึ่งใช้ Git เหมือนกัน ความรู้เรื่อง Git ในเล่มนี้จึงใช้กับบริการเหล่านั้นได้ทั้งหมด ส่วนที่เป็นหน้าจอของ GitHub โดยเฉพาะจะต่างกันบ้าง

1.4 Repository, commit และประวัติของโปรเจกต์

ก่อนลงมือในบทต่อไป มีศัพท์ที่จะเจอแทบทุกหน้าอยู่ไม่กี่คำ

- **Repository** หรือเรียกสั้น ๆ ว่า **repo** คือโฟลเดอร์โปรเจกต์ที่ Git ดูแลอยู่ หน้าตาเหมือนโฟลเดอร์ทั่วไป มีไฟล์ index.html, style.css, app.js อยู่ตามปกติ ที่ต่างคือข้างในมีโฟลเดอร์ซ่อนชื่อ .git ซึ่งเป็นที่เก็บประวัติทั้งหมด เราไม่ต้องเปิดหรือแก้อะไรในโฟลเดอร์นี้เอง Git จัดการให้
- **Commit** คือจุดบันทึกหนึ่งจุด ประกอบด้วยสภาพของไฟล์ทั้งโปรเจกต์ ณ ตอนนั้น ข้อความอธิบาย ชื่อและอีเมลของผู้บันทึก วันเวลา และรหัสประจำ commit ที่เป็นตัวอักษรและตัวเลขยาว ๆ เช่น 3f9c2ab ในบทที่ 3 เราจะเห็นว่ารหัสนี้ใช้อ้างถึง commit แต่ละจุดเวลาสั่งงาน Git
- **ประวัติ (history)** คือ commit ทั้งหมดเรียงต่อกัน commit ใหม่แต่ละจุดจำได้ว่าตัวเองต่อจาก commit ไหน ประวัติของแอปเมนูร้านกาแฟหลังจากทำงานไปสักพักอาจมีหน้าตาแบบนี้ เรียงจากใหม่ไปเก่า

```
a71d0e4  เปลี่ยนสีชื่อร้านเป็นสีเขียว

3f9c2ab  เพิ่มปุ่มเรียงตามราคา

9b04c1d  เพิ่มเมนูชาไทย

51e7a20  หน้าเมนูเริ่มต้น
```

แค่อ่านสี่บรรทัดนี้ก็รู้ว่าแอปพัฒนามาอย่างไร และถ้าแอปพังหลัง commit บนสุด ก็รู้ทันทีว่าควรเริ่มตรวจที่ไหน

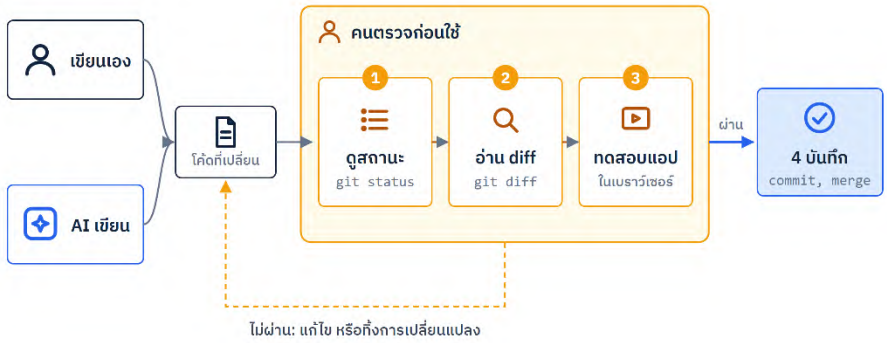
1.5 โค้ดที่เขียนเองและโค้ดจาก AI ต้องตรวจอะไรบ้าง

Git ช่วยเราได้มาก แต่ไม่ได้ทำให้โค้ดถูกต้อง Git บันทึกทุกอย่างที่เราสั่งให้บันทึก รวมถึงโค้ดที่พังด้วย commit จึงยืนยันได้แค่ว่า "ไฟล์เคยอยู่ในสภาพนี้" ไม่ได้ยืนยันว่า "แอปทำงานถูก" หน้าที่ตรวจว่าแอปถูกต้องยังเป็นของเรา

ข้อนี้สำคัญเป็นพิเศษในยุคที่ AI เขียนโค้ดได้เร็ว ผู้ช่วย AI ทำงานตามคำสั่ง แต่บ่อยครั้งทำเกินคำสั่ง เช่นจัดรูปแบบไฟล์ใหม่ทั้งไฟล์ เปลี่ยนชื่อตัวแปร หรือลบโค้ดที่มันคิดว่าไม่ได้ใช้ คำอธิบายที่ AI ตอบกลับมาก็อาจไม่ครบทุกจุดที่แก้ เหมือนเหตุการณ์ที่สองในหัวข้อ 1.1 ส่วนโค้ดที่เราเขียนเองก็พลาดได้ไม่ต่างกัน โดยเฉพาะตอนเหนื่อยหรือรีบ

หนังสือเล่มนี้จึงฝึกผู้อ่านให้รู้จักกับโค้ดทุกชิ้น ไม่ว่าใครเขียน เรียกว่าวงจร "ตรวจก่อนใช้" มี 4 ขั้น คือ

1. ดูสถานะ ว่ามีไฟล์ไหนเปลี่ยนบ้าง ตรงกับที่ตั้งใจหรือไม่
2. อ่าน diff ว่าแต่ละไฟล์เปลี่ยนไปอย่างไร มีบรรทัดที่ไม่เกี่ยวกับงานหรือไม่
3. ทดสอบแอปจริง เปิดในเบราว์เซอร์ กดปุ่ม ดูว่าฟีเจอร์ใหม่ทำงานและของเดิมไม่พัง
4. ผ่านทั้งสามข้อแล้วจึง commit หรือรวมงาน



ภาพที่ 1-3 วงจรตรวจก่อนใช้ ใช้กับโค้ดทุกชิ้นไม่ว่าใครเขียน

ขั้นตอน 4 ขั้นนี้ฟังดูยุ่งยากสำหรับการเปลี่ยนสีหัวข้อเพียงบรรทัดเดียว แต่ในทางปฏิบัติใช้เวลาไม่ถึงนาทีเมื่อคล่องแล้ว และคุ้มกว่ามากเมื่อเทียบกับการนั่งไล่หาว่าเมนูหายไปเพราะอะไรตอนตีสอง ในแต่ละบทจะมีกล่อง "ตรวจก่อนใช้" ที่เพิ่มรายละเอียดขึ้นทีละขั้น จากการดูสถานะในบทที่ 3 ไปจนถึงให้ GitHub ตรวจโค้ดอัตโนมัติในบทที่ 11

เคล็ดลับ

ตรวจก่อนใช้: ทุกครั้งที่ AI ตอบว่า "แก้เรียบร้อยแล้ว" ให้ถือว่าเป็นคำบอกเล่าที่ยังไม่ได้ยืนยัน หลักฐานจริงคือ diff และผลทดสอบในเบราว์เซอร์

1.6 แอปเมนูร้านกาแฟและเส้นทางของเล่มนี้

ตลอดทั้งเล่มเราจะพัฒนาแอปตัวเดียวต่อเนื่องกัน คือหน้าเว็บเมนูร้านกาแฟสมมติชื่อ "ร้านกาแฟบ้านสวน" เขียนด้วย HTML, CSS และ JavaScript ล้วน

มีสามไฟล์ เปิดด้วยเบราว์เซอร์ได้ทันทีโดยไม่ต้องติดตั้งเครื่องมืออื่น เราเลือกแอปขนาดเล็กแบบนี้เพื่อให้ความสนใจอยู่ที่ Git ไม่ใช่ที่ตัวภาษาโปรแกรม ใครเขียนเว็บไม่เป็นเลยก็ทำตามได้ เพราะทุกบทมีโค้ดเตรียมไว้ให้

ทุกแบบฝึกที่ต้องแก้โค้ดมีให้เลือกสองทาง ทางแรกคือแก้เองตามโค้ดตัวอย่าง ทางที่สองคือสั่งผู้ช่วย AI ตัวไหนก็ได้ด้วย prompt ภาษาไทยที่เตรียมไว้ ไม่ต้องสมัครบริการ AI แบบเสียเงิน ทั้งสองทางต้องได้แอปที่ทำงานตามเกณฑ์เดียวกัน แม้โค้ดและรหัส commit ของแต่ละคนจะไม่เหมือนกันก็ไม่เป็นไร

แอปจะเติบโตไปตามบทต่าง ๆ ดังตารางที่ 1-1

ตารางที่ 1-1 ฟีเจอร์ของแอปและทักษะ Git ที่ฝึกในแต่ละบท

บท	สิ่งที่เพิ่มในแอป	สิ่งที่ฝึกด้าน Git
3	ชื่อร้านและคำอธิบายเมนู	commit แรก
4	สีธีม	อ่าน diff จับการแก้เกินโจทย์
5	(แก้ของที่พัง)	ย้อนเวลาและกู้งาน
6	ธีมมืด	ทดลองบน branch
7	ราคาลาเต้	แก้ merge conflict
8	README	ส่งงานขึ้น GitHub
9	ตัวกรองเมนู	ใช้ AI แบบตรวจได้
10	หมวดขนม	Pull Request และรีวิว
11	ชุดทดสอบ และเว็บออนไลน์	GitHub Actions และ Pages
12	ตะกร้าสาธิต	ทำครบทุกขั้นจนออกเวอร์ชัน

โดยภายในเล่มแบ่งเป็น 4 ภาค ภาค 1 (บทที่ 1 ถึง 4) เริ่มบันทึกและตรวจโค้ดในเครื่อง ภาค 2 (บทที่ 5 ถึง 7) แก้พลาดและทดลองฟีเจอร์ ภาค 3 (บทที่ 8 ถึง 10) ขึ้น GitHub และทำงานร่วมกับ AI และคน ภาค 4 (บทที่ 11 และ 12) ตรวจอัตโนมัติ เผยแพร่ และส่งมอบงาน

ไฟล์ประกอบของทุกบท ทั้งโค้ดตั้งต้นและเฉลย ดาวน์โหลดได้จากหน้าเว็บประจำเล่มตามที่ระบุในส่วน "วิธีใช้หนังสือเล่มนี้" ต้นเล่ม

Workshop สืบหาสาเหตุที่เมนูหาย

แบบฝึกนี้ให้ลองทำหน้าที่ของ Git ด้วยมือ เพื่อให้เห็นว่าเครื่องมือที่จะเรียนในบทต่อไปช่วยประหยัดแรงตรงไหน ไม่ต้องติดตั้งอะไร ใช้แค่เบราว์เซอร์และโปรแกรมแก้ข้อความ

Workshop สืบหาสาเหตุที่เมนูหาย

1. แตกไฟล์ประกอบบทที่ 1 จะได้โฟลเดอร์ ch01-compare ข้างในมีโฟลเดอร์ before และ after แต่ละโฟลเดอร์มีไฟล์ index.html, style.css และ app.js
2. ดับเบิลคลิก index.html ในโฟลเดอร์ before เพื่อเปิดในเบราว์เซอร์ ควรเห็นชื่อร้านสีน้ำตาลและรายการเครื่องดื่มห้ารายการ กดปุ่ม "เรียงตามราคา" รายการจะเรียงจากชาไทย 40 บาทขึ้นไป
3. เปิด index.html ในโฟลเดอร์ after ควรเห็นชื่อร้านเป็นสีเขียว แต่ไม่มีรายการเครื่องดื่มเลย
4. เปิดไฟล์ style.css ของทั้งสองโฟลเดอร์คู่กัน หาบรรทัดที่ต่างกัน จดไว้ว่าอยู่บรรทัดไหนและเปลี่ยนจากอะไรเป็นอะไร
5. ทำแบบเดียวกันกับ app.js และ index.html

6. ตอบคำถาม: บรรทัดไหนคือการแก้ที่ตั้งใจ บรรทัดไหนทำให้เมนูหาย และถ้าจะให้ได้ชื่อร้านสีเขียวพร้อมเมนูที่ทำงานปกติ ต้องย้อนเฉพาะส่วนไหน

.....

.....

.....

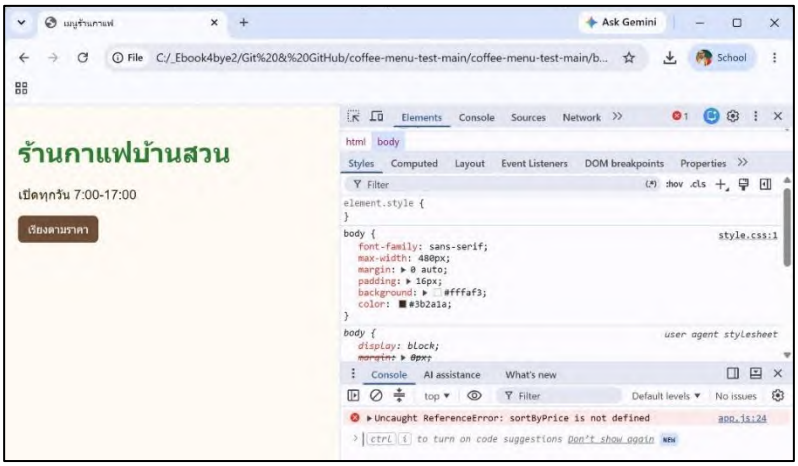
.....

.....

.....

.....

สำหรับคนที่ใช้ Chrome หรือ Edge มีวิธีหาสาเหตุเร็วกว่านั้น กดปุ่ม F12 ในหน้า after แล้วเลือกแท็บ Console จะเห็นข้อความสีแดงทำนองว่า sortByPrice is not defined พร้อมชื่อไฟล์และเลขบรรทัด ข้อความนี้บอกว่าโค้ดเรียกหาฟังก์ชันที่ไม่มีอยู่แล้ว เป็นเบาะแสที่ดี แต่ยังไม่บอกว่าฟังก์ชันนั้นหายไปเพราะอะไร คำตอบนั้นต้องได้จากการเทียบไฟล์



ภาพที่ 1-4 แท็บ Console (F12)

ส่วนที่สองของ workshop คือดูว่าโปรเจกต์จริงเก็บประวัติอย่างไร เปิดเบราว์เซอร์ไปที่ github.com/github/gitignore/commits/main ซึ่งเป็น repo

สาธารณะที่ GitHub ใช้รวบรวมแม่แบบไฟล์ .gitignore (เราจะได้ใช้ไฟล์ชนิดนี้ในบทที่ 4) หน้านี้เปิดดูได้โดยไม่ต้องมีบัญชี

Workshop อ่านประวัติของโปรเจกต์จริง

1. ในหน้ารายการ commit สังเกตว่าแต่ละแถวมีข้อความอธิบาย ชื่อผู้บันทึก วันที่ และรหัส commit สั้น ๆ ทางขวา
2. คลิกข้อความของ commit ใดก็ได้หนึ่งรายการ หน้าใหม่จะแสดงไฟล์ที่เปลี่ยนใน commit นั้น บรรทัดพื้นเขียวคือบรรทัดที่เพิ่ม บรรทัดพื้นแดงคือบรรทัดที่ลบ
3. เลือก commit ที่ข้อความบอกชัดว่าทำอะไร แล้วดูว่าสิ่งที่เปลี่ยนจริงตรงกับข้อความหรือไม่
4. ลองหา commit ที่ข้อความไม่ค่อยสื่อ แล้วคิดว่าถ้าเป็นคนเขียนเอง จะเขียนข้อความว่าอะไร

.....

.....

.....

.....

.....

.....

เฉลยของ workshop คือ ในโฟลเดอร์ after มีการแก้สองจุด คือสีหัวข้อใน style.css กับชื่อฟังก์ชันใน app.js และจุดที่ทำให้เมนูหายคือชื่อฟังก์ชัน ถ้าจะแก้ให้ย้อนเฉพาะชื่อฟังก์ชันกลับเป็นของเดิม และเก็บสีเขียวไว้ อีกเรื่องที่ต้องจำคือ ต่อให้มี commit เก็บไว้ทั้งก่อนและหลังแก้ commit ก็ไม่ได้บอกว่าแอปพัง คนที่ต้องเปิดเบราว์เซอร์ทดสอบคือเรา

สรุปท้ายบท

ถึงตรงนี้เราเปรียบเทียบไฟล์สองชุดด้วยมือและหาต้นเหตุที่เมนูหายได้ ซึ่งเป็นงานเดียวกับที่ Git จะทำให้ในคำสั่งเดียวตั้งแต่บทที่ 4 เราแยกได้ว่า Git เป็นโปรแกรมในเครื่องที่บันทึกประวัติ ส่วน GitHub เป็นบริการออนไลน์สำหรับฝากและร่วมงาน และรู้ว่าการ Save ไฟล์ต่างจากการ commit อย่างไร

ที่สำคัญกว่าคำสั่งใด ๆ คือวงจรตรวจก่อนใช้ ดูสถานะ อ่าน diff ทดสอบ แล้วจึงบันทึก ใช้กับโค้ดทุกชิ้นทั้งที่เขียนเองและที่ AI เขียน บทถัดไปจะติดตั้ง Git และ VS Code แล้วเปิดแอปเมนูร้านกาแฟในเครื่องของเราเอง

